Computing the Diameter of a Point Set

Grégoire Malandain, Jean-Daniel Boissonnat

International Journal of Computational Geometry & Applications

COMPUTING THE DIAMETER OF A POINT SET

GRÉGOIRE MALANDAIN
Epidaure team, INRIA, 2004 route des lucioles BP 93, 06 902 Sophia-Antipolis Cedex, France,
gregoire.malandain@sophia.inria.fr

and

JEAN-DANIEL BOISSONNAT
Prisme team, INRIA, 2004 route des lucioles BP 93, 06 902 Sophia-Antipolis Cedex, France,
jean-daniel.boissonnat@sophia.inria.fr

Received Received (received date)
Revised Revised (revised date)
Communicated by Communicated by (Name)

ABSTRACT

Given a finite set of points $\mathcal{P}$ in $\mathbb{R}^d$, the diameter of $\mathcal{P}$ is defined as the maximum distance between two points of $\mathcal{P}$. We propose a very simple algorithm to compute the diameter of a finite set of points. Although the algorithm is not worst-case optimal, an extensive experimental study has shown that it is extremely fast for a large variety of point distributions. In addition, we propose a comparison with the recent approach of Har-Peled [5] and derive hybrid algorithms to combine advantages of both approaches.

Keywords: diameter, width, approximation, point sets, furthest neighbors, double normal.

1. Introduction

Given a set $\mathcal{P}$ of n points in $\mathbb{R}^d$, the *diameter* of $\mathcal{P}$ is the maximum Euclidean distance between any two points of $\mathcal{P}$.

Computing the diameter of a point set has a long history. By reduction to set disjointness, it can be shown that computing the diameter of n points in $\mathbb{R}^d$ requires $\Omega(n \log n)$ operations in the algebraic computation-tree model.[7] A trivial $O(n^2)$ upper-bound is provided by the brute-force algorithm that compares the distances between all pairs of points. It has been noted by Yao that it is possible to compute the diameter in sub-quadratic time in any dimension.[11] In dimensions 2 and 3, better solutions are known. In the plane, the problem can easily be solved in optimal $O(n \log n)$ time. The problem becomes much harder in $\mathbb{R}^3$. Clarkson and Shor gave a randomized $O(n \log n)$ algorithm.[4] This algorithm involves the computation of the intersection of n balls (of the same radius) in $\mathbb{R}^3$ and the fast location of points with respect to this intersection. This makes the algorithm difficult to

implement and presumably slower in practice than the brute-force algorithm for most practical data sets. Moreover this algorithm is not efficient in higher dimensions since the intersection of n balls of the same radius has size $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$. Recent attempts to solve the 3-dimensional diameter problem led to $O(n \log^3 n)$ [1,8] and $O(n \log^2 n)$ deterministic algorithms.[9,2] Finally Ramos found an optimal $O(n \log n)$ deterministic algorithm.[10] Cheong at al. have also proposed a $O(n \log^2 n)$ randomized algorithm that solves the all-pairs farthest neighbor problem for n points in convex position in $\mathbb{R}^3$ [3]. As Bespamyatnikh's approach, their algorithm does not need to compute intersections of spheres.

All these algorithms use complex data structures and algorithmic techniques such as 3-dimensional convex hulls, intersection of balls, furthest-point Voronoi diagrams, point location search structures or parametric search. We are not aware of any implementation of these algorithms. We also suspect that they would be very slow in practice compared to the brute-force algorithm, even for large data sets.

Some of these algorithms could be extended in higher dimensions. However, this is not worth trying since the data structures they use have sizes that depend exponentially on the dimension: e.g. the size of the convex hull of n points or of the intersection of n balls of $\mathbb{R}^d$ can be as large as $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$.

Our algorithm works in any dimension. Moreover, it does not construct any complicated data structure; in particular, it does not require that the points are in convex position and does not require to compute the convex hull of the points. The only numerical computations are dot product computations as in the brute-force algorithm.

The algorithm is not worst-case optimal but appears to be extremely fast under most circumstances, the most noticeable exception occurring when the points are distributed on a domain of constant width, e.g. a sphere. We also propose an approximate algorithm.

Independently, Har-Peled has designed an algorithm which is also sensitive to the "hardness" of computing the diameter of the given input, and for most inputs is able to compute the exact diameter or an approximation very fast.[5] We compare both methods and also show that they can be combined so as to take advantage of the two.

2. Definitions, notations and geometric preliminaries

We denote by n the number of points of $\mathcal{P}$, by h the number of vertices of the convex hull of $\mathcal{P}$, and by D the diameter of $\mathcal{P}$. $\delta(\cdot,\cdot)$ denotes the Euclidean distance, and $\delta^2(\cdot,\cdot)$ the squared Euclidean distance.

The length of a pair of points pq of $\mathcal{P}$ is the Euclidean distance $\delta(p,q)$ between p and q. A pair of length D is called *maximal*.

For $p \in \mathcal{P}$, $FP(p)$ denotes the subset of the points of $\mathcal{P}$ that are furthest from p. Clearly,

$$q \in FP(p) \quad \Longleftrightarrow \quad q = \arg\max_{s \in \mathcal{P}} \overrightarrow{sp}.\overrightarrow{sp}$$

The pair pq is called a *double normal* if $p \in FP(q)$ and $q \in FP(p)$. If pq is a maximal pair, pq is a double normal. The converse is not necessarily true.

Observe that the points of a maximal pair or of a double normal belong to the convex hull of $\mathcal{P}$. Observe also that, if the points are in *general position*, *i.e.* there are no two pairs of points at the same distance, the number of double normals is at most $\lfloor h/2 \rfloor$.

$B(p,r)$ denotes the closed ball of radius r centered at p, $\Sigma(p,r)$ its bounding sphere. The ball with diameter pq is denoted by $B[pq]$ and its bounding sphere by $\Sigma[pq]$.

Since the distance between any two points in $B[pq]$ is at most $\delta(p,q)$, we have:

Lemma 1 *If $p, q \in \mathcal{P}$ and if pq is not a maximal pair, any maximal pair must have at least one point outside $B[pq]$.*

As a corollary, we have:

Lemma 2 *If $p, q \in \mathcal{P}$ and if $\mathcal{P} \setminus B[pq] = \emptyset$, pq is a maximal pair of $\mathcal{P}$ and $\delta(p,q)$ is the diameter of $\mathcal{P}$.*

3. Computation of a double normal

Algorithm 1 below repeatedly computes a furthest neighbor q of a point p of $\mathcal{P}$ until a double normal DN is found. To find a furthest neighbor of $p \in \mathcal{P}$, we simply compare p against all the other points in $\mathcal{P}$, and keep the one yielding the largest distance (this procedure is called a FP scan, and is detailed in Algorithm 1). Point p is then removed from $\mathcal{P}$ and won't be considered in further computations.

In this algorithm, as in the following ones, distances can be replaced by squared distances, thus avoiding square root extractions.

```
1: procedure DoubleNormal( p, P )    // p is a point of P
2: Δ := 0
3: repeat    // FP scan
4:    P := P \ {p}    // remove p from P for any further computation
5:    find q ∈ FP(p), i.e. one of the furthest neighbors of p
6:    if δ(p,q) > Δ then
7:       Δ := δ(p,q) and DN := pq
8:       p := q
9: until Δ stops increasing
10: return DN
```

Alg. 1. Computes a double normal.

Lemma 3 *Algorithm 1 terminates and returns a double normal.*

Proof. Δ can only take a finite number of different values and strictly increases: this ensures that the algorithm terminates. After termination we have $q \in FP(p)$ and all the points of $\mathcal{P}$ belong to $B(p, \delta(p,q))$. Since Δ has the same value the iteration before termination, all the points of $\mathcal{P}$ belong also to $B(q, \delta(p,q))$ and therefore $p \in FP(q)$. □

After termination of Algorithm 1, the original set $\mathcal{P}$ has been replaced by a strict subset $\mathcal{P}'$ since some points have been removed from $\mathcal{P}$ (line 4 of Algorithm 1). By

construction, the returned pair pq is a double normal of the original set $\mathcal{P}$. Moreover, we have

$$\delta(x,p) \leq \delta(p,q) \quad \text{and} \quad \delta(x,q) \leq \delta(p,q) \quad \forall x \in \mathcal{P}' .$$

Lemma 4 *Algorithm 1 performs at most h FP scans and takes $\Theta(nh)$ time.*

Proof.
The upper bound is trivial since all the points q that are considered by Algorithm 1 belong to the convex hull of $\mathcal{P}$ and all points q are distinct. As for the lower bound, we give an example in the plane, which is sufficient to prove the bound. Consider a set of $2n+1$ points $p_0, \ldots, p_{2n}$ placed at the vertices of a regular polygon P (in counterclockwise order). For $i > 0$, we slightly move the p_i outside P along the ray Op_i by a distance ε^i for some small $\varepsilon < 1$. Let p'_i be the perturbed points (see figure 1). It is easy to see that the furthest point from p'_i is always $p'_{i+n \bmod (2n+1)}$ except for p'_{n+1}. Therefore, the algorithm will perform FP scans starting successively at $p_{\sigma_0}, \ldots, p_{\sigma_{2n+1}}$ where $\sigma_i = i \times n$ (modulo $2n+1$). □

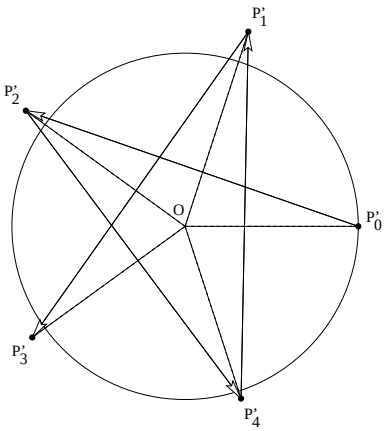

Fig. 1. Proof of lemma 4.

Although tight in the worst-case, the bound in lemma 4 is very pessimistic for many point distributions. This will be corroborated by experimental results.

4. Iterative computation of double normals

Assume that Algorithm 1 has been run and let $\mathcal{Q} = \mathcal{P} \setminus B[pq]$. If $\mathcal{Q} = \emptyset$, pq is a maximal pair and $\delta(p,q)$ is the diameter of $\mathcal{P}$ (lemma 2). Otherwise, we have to determine whether pq is a maximal pair or not. Towards this goal, we try to find a better (*i.e.* longer) double normal by running Algorithm 1 again, starting at a point in $\mathcal{Q}$ rather than in $\mathcal{P}$, which is sufficient by lemma 1. Although any point in $\mathcal{Q}$ will be fine, experimental evidence has shown that choosing the point m furthest from $\frac{p+q}{2}$ is usually better (m lies outside $B[pq]$ since $\mathcal{Q} \neq \emptyset$). We have

$$m = \arg\max_{s \in \mathcal{P}} \overrightarrow{sp}.\overrightarrow{sq} .$$

while points of $\mathcal{Q}$ are characterized by

$$\mathcal{Q} = \left\{ s \in \mathcal{P} \mid \overrightarrow{sp}.\overrightarrow{sq} > 0 \right\} .$$

It follows that

$$\mathcal{Q} = \emptyset \iff \overrightarrow{mp}.\overrightarrow{mq} \leq 0.$$

Therefore we simply need to scan $\mathcal{P}$ to determine whether $\mathcal{Q}$ is empty or not, and, if not, to find m (this procedure is called a DN scan). In our implementation, $\mathcal{P}$ is

stored in a list $\mathcal{L}$ of points, and the above scans consists in putting all the points of $\mathcal{Q}$ at the beginning of the list, so that $\mathcal{L} = \{\mathcal{Q}, \mathcal{P} \setminus \mathcal{Q}\}$.

Algorithm 2 below repeats this procedure further until either $\mathcal{Q}$ becomes empty or the current maximal distance Δ does not increase.

```
1: Δ := 0   stop := 0
2: pick a point m ∈ P
3: repeat   // DN scan
4:    pq = DoubleNormal( m, P )   // double normal pq of length δ(p,q)
5:    if Δ < δ(p,q) then
6:       Δ := δ(p,q) and DN := pq
7:       Q := P \ B[pq]
8:       if Q ≠ ∅ then
9:          find m ∈ Q a furthest point from (p+q)/2
10:   else
11:      stop := 1   // terminates with Q ≠ ∅.
12: until Q = ∅ or stop = 1
13: return DN := pq, Δ := δ(p,q)
```

Alg. 2. Iterated search for double normals.

Lemma 5 *Algorithm 2 performs $O(h)$ DN scans. Its overall time-complexity is $O(nh)$.*

Proof. The first part of the lemma comes from the fact that the algorithm enumerates (possibly all) double normals by strictly increasing lengths.

Let us prove now the second part of the lemma. Each time Algorithm 1 performs a FP scan starting at a point p (loop 3-9), p is removed from further consideration (line 4). Moreover, except for the first point p to be considered, all these points belong to the convex hull of $\mathcal{P}$. It follows that the total number of FP scans is at most $h + 1$. Since each FP scan takes $O(n)$ time, we have proved the lemma. □

5. Diameter computation

Assume that Algorithm 2 terminates after I iterations. Since, at each iteration, a new double normal is computed, the algorithm has computed I double normals, noted $p_i q_i$, $i = 1, \ldots, I$, and we have $\delta(p_1, q_1) < \ldots < \delta(p_{I-1}, q_{I-1})$. We also have $\delta(p_{I-1}, q_{I-1}) < \delta(p_I, q_I)$ if Algorithm 2 stops with $\mathcal{Q} = \emptyset$.

Lemma 6 *The double normals $p_i q_i$, $i = 1, \ldots, J$*

$$\begin{array}{lll} J = I - 1 & \textit{if} & \delta(p_{I-1}, q_{I-1}) \geq \delta(p_I, q_I) \\ J = I & \textit{if} & \delta(p_{I-1}, q_{I-1}) < \delta(p_I, q_I) \end{array}$$

computed in Algorithm 2 are also double normals of the original data set $\mathcal{P}$.

Proof. Each time Algorithm 1 is called, some points are removed from the original data set. We rename the original data set $\mathcal{P}^{(0)}$ and denote by $\mathcal{P}^{(i)}$ the set of points that remain after the i-th iteration, *i.e.* the one that computes $p_i q_i$. Hence each set $\mathcal{P}^{(i)}$, $i > 0$, is strictly included in $\mathcal{P}^{(i-1)}$.

By construction, each pair p_iq_i, $i = 1, \ldots, I$, is a double normal for $\mathcal{P}^{(i-1)}$. Because $\mathcal{P}^{(0)} \supset \cdots \supset \mathcal{P}^{(i)} \supset \cdots \supset \mathcal{P}^{(I)}$, it is easily seen that each pair p_iq_i, $i = 1, \ldots, I$, verifies

$$\delta(x, p_i) < \delta(p_i, q_i) \quad \text{and} \quad \delta(x, q_i) < \delta(p_i, q_i) \quad \forall x \in \mathcal{P}^{(j)} \quad j = i-1, \ldots, I\,. \quad (1)$$

Now, consider all the pairs p_iq_i, $i = 1, \ldots, J$, with

$$\begin{array}{lll} J = I - 1 & \text{if} & \delta(p_{I-1}, q_{I-1}) \geq \delta(p_I, q_I) \\ J = I & \text{if} & \delta(p_{I-1}, q_{I-1}) < \delta(p_I, q_I) \end{array}$$

We have

$$\delta(p_1, q_1) < \cdots < \delta(p_i, q_i) < \cdots < \delta(p_J, q_J)$$

Let x be a point that has been removed (line 4 of Algorithm 1) from the original point set $\mathcal{P}$ before the computation of p_iq_i (*i.e.* $x \in \mathcal{P} \setminus \mathcal{P}^{(i-1)}$). Prior to its removal, x has been compared to all other points, including p_i and q_i. This implies that $\delta(x, p_i) < \delta(x, FP(x))$. Moreover, by construction we have $\delta(x, FP(x)) \leq \delta(p_x, q_x)$ where p_xq_x is the double normal found at the iteration where x has been removed from $\mathcal{P}$. Finally we have $\delta(p_x, q_x) < \delta(p_i, q_i)$. We conclude that

$$\delta(x, p_i) < \delta(p_i, q_i) \quad \forall x \in \mathcal{P} \setminus \mathcal{P}^{(i-1)} \quad (2)$$

and the same inequality holds for q_i. Together, Equations 1 and 2 show that p_iq_i is a double normal of the original point set $\mathcal{P}$. □

When Algorithm 2 terminates, we are in one of the two following cases :

Case 1 : $\delta(p_I, q_I) > \delta(p_{I-1}, q_{I-1})$ and $\mathcal{Q} = \mathcal{P}^{(I)} \setminus B[p_Iq_I] = \emptyset$.

Here, by lemma 6, p_Iq_I is a double normal of $\mathcal{P}$, and by lemma 2, it is a maximal pair of $\mathcal{P}$.

Case 2 : $\delta(p_I, q_I) \leq \delta(p_{I-1}, q_{I-1})$.

In this case, $\mathcal{P}^{(I-1)} \setminus B[p_{I-1}\, q_{I-1}]$ was not empty *before* the computation of $[p_I, q_I]$. We have to determine whether $p_{I-1}q_{I-1}$ is a maximal pair or not. Thanks to lemma 1, if a longer double normal exists, one of its endpoints lies in $\mathcal{P}^{(I)} \setminus B[p_{I-1}\, q_{I-1}]$. If this last set is empty, which is checked by Algorithm 3, $p_{I-1}q_{I-1}$ is a maximal pair of $\mathcal{P}$.

Required: $\mathcal{P}^{(I)}$ and $p_{I-1}q_{I-1}$ (provided by Algorithm 2)
1: $\mathcal{Q} := \mathcal{P}^{(I)} \setminus B\,[p_{I-1}\, q_{I-1}]$
2: **if** $\mathcal{Q} = \emptyset$ **then**
3: $\quad p_{I-1}q_{I-1}$ is a maximal pair of $\mathcal{P}$

Alg. 3. Checks whether $\mathcal{Q} = \mathcal{P}^{(I)} \setminus B[p_{I-1}\, q_{I-1}] = \emptyset$.

If $\mathcal{Q} = \mathcal{P}^{(I)} \setminus B[p_{I-1}\, q_{I-1}] \neq \emptyset$, we have to check whether there exists a maximal pair with a point in this set. To search for such maximal pairs, we propose two methods. For clarity, we will write $\mathcal{P}$ instead $\mathcal{P}^{(I)}$ in the following.

Required: Δ (provided by Algorithm 2) and $\mathcal{Q}$ (provided by Algorithm 3)
1: **if** $\mathcal{Q} \neq \emptyset$ **then** // `Exhaustive search with an endpoint in` $\mathcal{Q}$
2: **for all** points $p_i \in \mathcal{Q}$ **do**
3: **for all** points $p_j \in \mathcal{P}$ **do**
4: **if** $\delta(p_i, p_j) > \Delta$ **then**
5: $\Delta := \delta(p_i, p_j)$
6: **return** Δ

Alg. 4. Exhaustive search over $\mathcal{Q} \times \mathcal{P}$.

5.1. Exhaustive search over $\mathcal{Q} \times \mathcal{P}$

The first method (Algorithm 4) simply considers all pairs in $\mathcal{Q} \times \mathcal{P}$.

5.2. Reduction of $\mathcal{Q}$

As it might be expected and is confirmed by our experiments (see section 7), the observed total complexity is dominated by the exhaustive search of the previous section (Algorithm 4). It is therefore important to reduce the size of $\mathcal{Q}$. For that purpose, we propose to reuse all the computed pairs $p_i q_i$, $i = 1, \ldots, I-2$, and $p_I q_I$.

5.2.1. Principle

Assume that we have at our disposal an approximation Δ of the diameter of $\mathcal{P}$ and a subset $\mathcal{Q} \subset \mathcal{P}$ that contains at least one endpoint of each maximal pair longer than Δ (plus possibly other points). To identify such endpoints in $\mathcal{Q}$ (*i.e.* to find the maximal pairs longer than Δ), we may, as in Algorithm 4, exhaustively search over $\mathcal{Q} \times \mathcal{P}$. The purpose of this section is to show how this search can be reduced.

Under the assumption that the diameter of $\mathcal{P}$ is larger than Δ, any maximal pair has at least one point outside any ball of radius $\Delta/2$.

Consider such a ball B' of radius $\Delta/2$. The exhaustive search over $\mathcal{Q} \times \mathcal{P}$ can then be reduced to two exhaustive searches associated to a partition of $\mathcal{Q}$ into $\mathcal{Q} \cap B'$ and $\mathcal{Q} \setminus B'$. More precisely, if $p \in \mathcal{Q}$, searching for a point q such that $\delta(p, q) > \Delta$ reduces to searching q in $\mathcal{P} \setminus \mathcal{Q} \setminus B'$ if p belongs to B', and searching q in $\mathcal{P}$ otherwise.

This way, instead of searching over $\mathcal{Q} \times \mathcal{P}$, we search over $(\mathcal{Q} \cap B') \times (\mathcal{P} \setminus \mathcal{Q} \setminus B')$ and $(\mathcal{Q} \setminus B') \times \mathcal{P}$, therefore avoiding searching a maximal pair in $(\mathcal{Q} \cap B') \times (\mathcal{P} \cap B')$.

B' should be chosen so as to maximize the number of points in $\mathcal{P} \cap B'$, which reduces the cost of searching over $(\mathcal{Q} \cap B') \times (\mathcal{P} \setminus \mathcal{Q} \setminus B')$. The idea is to reuse the already found pairs $p_i q_i$ (that are double normals of $\mathcal{P}$) and to iteratively center the balls of radius $\Delta/2$ at the points $\frac{p_i + q_i}{2}$.

In our implementation, $\mathcal{P}$ is still stored in a list $\mathcal{L}$ already partially ordered such that $\mathcal{L} = \{\mathcal{Q}, \mathcal{P} \setminus \mathcal{Q}\}$. Given some ball B', each sublist of $\mathcal{L}$ is also partially ordered so that $\mathcal{L} = \{\mathcal{Q} \setminus B', \mathcal{Q} \cap B', \mathcal{P} \setminus \mathcal{Q} \setminus B', \mathcal{P} \setminus \mathcal{Q} \cap B'\}$.

5.2.2. Algorithm

Assume that Algorithm 2 terminates under case 2, yielding the pair $p_{max} q_{max}$ (*i.e.* $p_{I-1} q_{I-1}$) of length $\Delta = \delta(p_{max}, q_{max})$ which is considered as an estimation

Required: $\Delta := \delta(p_{max}, q_{max})$ and $\mathcal{S}$ provided by Algorithm 2
Required: $\mathcal{Q}^{(0)} := \mathcal{Q}$ provided by Algorithm 3
1: **for all** pairs $p_i q_i \in \mathcal{S}$, $i := 1 \dots |S|$ **do**
2: $B' := B\left(\frac{p_i+q_i}{2}, \Delta/2\right)$
3: $d := \max \delta(p,q) \quad (q,p) \in \left(\mathcal{Q}^{(i-1)} \cap B'\right) \times \left(\mathcal{P} \setminus \mathcal{Q}^{(i-1)} \setminus B'\right)$
4: **if** $d > \Delta$ **then** // `A better diameter estimation was found`
5: $\Delta := d$
6: Add pair pq to set $\mathcal{S}$
7: $\mathcal{Q}^{(i)} := \mathcal{Q}^{(i-1)} \setminus B'$ // `new set` $\mathcal{Q}$
8: **if** $\mathcal{Q}^{(i)} = \emptyset$ **then**
9: return Δ // `diameter has been found`

Alg. 5. Iterative reduction of $\mathcal{Q}$ by successive examination of all pairs $p_i q_i$.

of the diameter. Moreover, we assume that the set $\mathcal{Q}$ computed by Algorithm 3 is not empty.

All the double normals $p_i q_i$ that have been found by Algorithm 2, except $p_{I-1}q_{I-1}$, are collected into a set $\mathcal{S}$.

If Algorithm 5 terminates with $\mathcal{Q}^{(|S|)} \neq \emptyset$, one still must run Algorithm 4 with $\mathcal{Q} = \mathcal{Q}^{(|S|)}$, *i.e.* the exhaustive search over $\mathcal{Q}^{(|S|)} \times \mathcal{P}$.

6. Diameter approximation

The length of a double normal is a lower bound for the diameter. Hence our algorithm provides a lower bound $\Delta \stackrel{\text{def}}{=} \Delta_{\min}$ on the diameter. We also get a trivial upper bound $\Delta_{\max} = \Delta_{\min}\sqrt{3}$. Indeed, let pq be the double normal whose length is $\Delta_{\min} = \delta(p,q)$. All the points of $\mathcal{P}$ belong to the intersection of the two balls of radius $\Delta_{\min}$ centered at p and q.

With only slight modifications, our algorithm can also be used to check whether this lower bound $\Delta_{estimate} = \Delta_{\min}$ is a ε-approximation of the true diameter Δ_{true}, *i.e.* verifies

$$\Delta_{estimate} \leq \Delta_{true} \leq (1+\varepsilon)\Delta_{estimate}$$

Indeed, we may use the point m furthest from $\frac{p+q}{2}$ that has been computed at line 9 of Algorithm 2. The distance from m to $\frac{p+q}{2}$ is given by

$$\delta^2\left(m, \frac{p+q}{2}\right) = \overrightarrow{mp}.\overrightarrow{mq} + \frac{\delta^2(p,q)}{4}\,.$$

An upper bound on the diameter is obviously $2\delta\left(m, \frac{p+q}{2}\right)$. Therefore if

$$2\delta\left(m, \frac{p+q}{2}\right) \leq (1+\varepsilon)\delta(p,q) \quad \Longleftrightarrow \quad 1 + 4\,\frac{\overrightarrow{mp}.\overrightarrow{mq}}{\delta^2(p,q)} \leq (1+\varepsilon)^2 \qquad (3)$$

we get the desired upper bound: $\Delta_{true} \leq (1+\varepsilon)\Delta_{estimate} = (1+\varepsilon)\delta(p,q)$.

If we look for an ε-approximation of the diameter, Algorithm 2 has to stop as soon as Condition (3) is satisfied. As for the computation of the exact diameter (section 4) we compute $m = \arg\max_{s \in \mathcal{P}} \overrightarrow{sp}.\overrightarrow{sq}$.

If $\overrightarrow{mp}.\overrightarrow{mq} \leq 0$, then $\mathcal{P} \setminus B[pq] = \emptyset$ and pq is the true diameter

else if inequality (3) is verified, pq is even a ε'-approximation of the diameter with $\varepsilon' = \sqrt{1 + 4\,\overrightarrow{mp}.\overrightarrow{mq}/\delta^2(p,q)} - 1$

else m is used to find another double normal.

Another modification has also to be done. The diameter of balls used in Algorithms 3 and 5 has to be now $\Delta(1+\varepsilon)$ instead of Δ. Thus line 1 of Algorithm 3 has to be replaced by $\mathcal{Q} := \mathcal{P}^{(I)} \setminus B\left(\frac{p_{I-1}+q_{I-1}}{2}, \frac{\Delta(1+\varepsilon)}{2}\right)$ and the ball B' used line 2 of Algorithm 5 has to be defined by $B' := B\left(\frac{p_i+q_i}{2}, \frac{\Delta(1+\varepsilon)}{2}\right)$.

7. Experimental results

In this section, we present experimental results with both the exact and the approximate algorithms. In all cases, we run the method with and without the reduction of $\mathcal{Q}$ described in section 5.2.

7.1. Point distributions in $\mathbb{R}^d$

In our experiments, we use several point distributions. For each of them, the limit value of the diameter (for an infinite number of points) is 1. O denotes the origin.

Volume based distributions

Points in a cube. The points are chosen independently according to a uniform distribution in $\left[-1/\left(2\sqrt{d}\right), 1/\left(2\sqrt{d}\right)\right]^d$.

For a large number of points, the maximal pair of the point set is close to a diagonal of the cube. The number of diagonals of the cube is 2^{d-1} and therefore increases dramatically with the space dimension.

Points in a ball. The points are chosen independently according to a uniform distribution in the ball of radius 1/2 centered at O.

Surface based distributions

Points on a sphere. The points are chosen independently according to a uniform distribution on the sphere of radius 1/2 centered at O.

Points on an ellipsoid. The points are chosen independently according to a uniform distribution on the surface of an ellipsoid centered at O. The ellipsoid axes are aligned with the coordinate axes.

The largest axis of the ellipsoid is set to 1/2 while the others are chosen independently according to a uniform distribution in $[1/10, 1/2]$.

Points on a gentle ellipsoid. The points are chosen independently according to a uniform distribution on the surface of an ellipsoid centered at O. The ellipsoid axes are aligned with the coordinate axes.

The largest axis $r_0 = r_{max}$ is set to 1/2, the smallest one $r_{n-1} = r_{min}$ is set to 1/10, while the other axes r_i for $i = 1, \ldots, (n-2)$, are chosen independently according to a uniform distribution in

$$\left[r_{max} - \left(i + \frac{1}{2} \right) \frac{r_{max} - r_{min}}{n-1}, r_{max} - \left(i - \frac{1}{2} \right) \frac{r_{max} - r_{min}}{n-1} \right].$$

Notice that the axes are taken from non-overlapping intervals.

These ellipsoids cannot have an infinite number of maximal double normals. When the number of points is large enough, the maximal pair of the point set is close to the main axis of the ellipsoid.

7.2. Performance measure

The only numerical operations used by our algorithm are dot products. The complexity of the proposed algorithm can then be empirically estimated by simply counting the number of dot products performed to compute the diameter or its approximation.

Below, we use as a measure of performance the number of dot products divided by the number n of points in $\mathcal{P}$. As an example, the complexity of the brute-force algorithm that considers all $n(n-1)/2$ pairs is $n(n-1)/2$ (as $n(n-1)/2$ dot products are computed), and its performance is $(n-1)/2$.

Interestingly, the computation of a double normal, and the iterative search for double normals are always very fast and exhibit (at least experimentally) a linear complexity. Section 7.3 will present empirical results. The overall complexity and computing time are dominated by the exhaustive examination of all pairs of $\mathcal{Q} \times \mathcal{P}$.

When provided, CPU times (in seconds) are obtained on a PC running linux (Compaq professional workstation AP550, biprocessor, 866 MHz). The C code[a] is compiled with the `gcc` compiler (version 2.95.3) and the `-O` option.

7.3. Double normals and iterative search for double normals

It is of some interest to estimate the complexity of the first part of the algorithm, *i.e.* the iterative search for double normals presented in Algorithm 2.

Table 1. Number of FP scans (Algorithm 1) and DN scans (Algorithm 2). For each point distribution, we give the measured minimum, maximum and average numbers of scans.

	3-D points distributions							
	Volume				Surface			
	cube		sphere		ellipsoid		sphere	
# FP scan	2 - 5	2.12	2 - 7	2.39	2 - 103	6.01	2 - 6	2.20
# DN scan	1 - 5	2.50	2 - 7	2.86	1 - 8	1.98	2 - 7	2.65

According to our experiments (with n ranging from 10,000 to 100,000), these figures do not depend on n but only on the point distribution. Thus, for each point

[a] Available at `http://www-sop.inria.fr/epidaure/personnel/malandain/diameter/`.

distribution we present the minimum, the maximum, and the average numbers of scans. As a consequence, the (empirical) complexity of this part of the algorithm appears to be $O(n)$.

A last noticeable remark has to be done concerning Table 1: a large number of FP scans may be needed to compute a double normal for points distributed on an ellipsoid. This case is similar to the worst-case example described in the proof of lemma 4. It will be further described in section 7.4.2.

7.4. Point sets in $\mathbb{R}^3$

7.4.1. Exact diameter

Table 2. Performance of the computation of the exact diameter without and with reduction of $\mathcal{Q}$ for various 3-D point distributions. Measures are averaged over several trials (mostly 500, sometimes 100 or 50).

	3-D Points distribution				
	Volume		Surface		
n	cube	sphere	gentle ellipsoid	ellipsoid	sphere
	Method with no reduction of $\mathcal{Q}$				
20,000	10.21	581.98	6.84	24.45	7514.7
40,000	9.92	849.87	6.98	25.57	15031.7
60,000	9.95	1210.21	7.27	26.58	22523.9
80,000	10.03	1378.34	7.09	35.45	30033.8
100,000	9.91	1729.55	7.30	36.25	37538.3
1,000,000	9.31	-	7.92	37.82	-
	Method with reduction of $\mathcal{Q}$				
20,000	9.52	60.22	6.91	13.43	4303.6
40,000	9.32	82.09	6.98	13.06	8493.0
60,000	9.36	88.28	7.19	17.60	12647.1
80,000	9.24	101.17	7.06	16.33	16857.6
100,000	9.38	126.78	7.33	15.98	21069.4
1,000,000	9.39	-	7.83	20.11	-

Table 2 gives the performance measure (see section 7.2) of the exact computation of the diameter without and with reduction of the set $\mathcal{Q}$ for various 3-D points distributions. It should be compared to the performance measure of the brute-force method, *i.e.* $(n-1)/2$. CPU times for the same experiments can be found in Table 9.

Several remarks can be made.

- The algorithm performs quite well (it is almost linear) for the cube and the gentle ellipsoid, and the reduction of $\mathcal{Q}$ does not improve (but does not deteriorate) the performances. It is due to the fact that, after the iterative search for a best double normal pq (Algorithm 2), only a very few points (sometimes none) are outside $B[pq]$.
- The algorithm performs a little bit worse for the general ellipsoid, with a more chaotic behavior. Typically the number of FP scans needed to compute a

double normal can be large. See the discussion in section 7.4.2. The reduction of $\mathcal{Q}$ improves somehow the performance.

- The performances are also quite bad for points distributed inside or on a sphere. This is due to the fact that, after the iterative search for a best double normal pq, there are respectively about 50% (about $n/2$) and about $n^{3/4}$ (empirical estimation) of the points of $\mathcal{P}$ outside $B[pq]$ for points distributed on the sphere and inside the sphere. In the case of points distributed on a sphere, the performances are close (but still a little bit better) to those of the brute-force method.

  It can also be seen that the reduction of $\mathcal{Q}$ improves the performance quite a lot when the points are distributed inside the sphere, but not that much when they lie on the sphere. Both empirical complexities appears to be quadratic.

  More results about sets of constant width (2D Reuleaux polygons) can be found in a previous research report.[6]

7.4.2. The ellipsoid case (in $\mathbb{R}^3$)

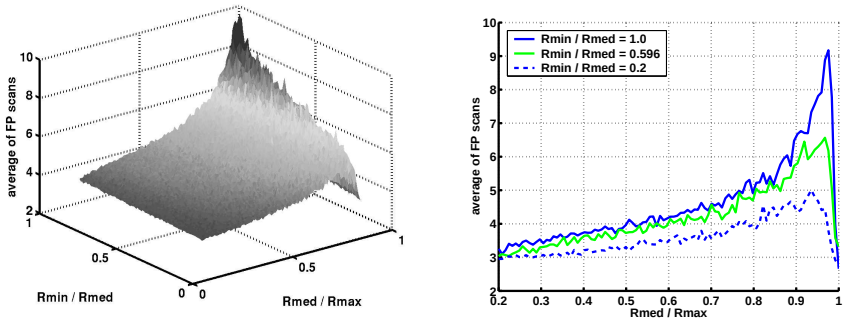

Fig. 2. Average number of FP scans needed to compute a double normal for points distributed on an ellipsoid, with respect to the axis length ratio. Experiments are conducted with $n = 10,000$.

The fact that the complexity varies widely for point distributed on both types of ellipsoids can be explained using more detailed experiments. Instead of randomly picking the axes of the ellipsoids, we let them vary continuously. More precisely, the largest axis $R_{\max}$ being set to 1/2 (see section 7.1), we study the behavior of our method with respect to the ratio $R_{\text{med}}/R_{\max}$ and $R_{\min}/R_{\text{med}}$ where $R_{\min}$ and R_{med} are respectively the smallest and the medium axes.

We observe that, for almost all $R_{\text{med}}/R_{\max}$ and $R_{\min}/R_{\text{med}}$, the number of points outside $B[pq]$ (pq being the best double normal computed by Algorithm 2) remains small and comparable to what we get in the case of the gentle ellipsoid. The only exception is for points on a sphere (*i.e.* when the three axes are equal) and when $R_{\text{med}}/R_{\max}$ is close to 1.

Interestingly, the average number of FP scans needed to compute a double normal increases with the ratio $R_{\text{med}}/R_{\max}$ (see Figure 2), but falls down when

$R_{\text{med}}/R_{\text{max}} = 1$. This is related to the construction described in the proof of lemma 4 where we had points close but not on a circle, leading to a large number of FP scans. This is exactly what we observe here, and this explains also the large maximum number of FP scans presented in Table 1.

7.4.3. Diameter approximation

Table 3. Performance measures for the computation of a 0.01-approximation of the diameter without and with reduction of $\mathcal{Q}$ for various 3-D point distributions. Measures are averaged of several trials (mostly 500, sometimes 100 or 50). It should be compared to Table 2.

	3-D Points distribution				
	Volume		Surface		
n	cube	sphere	gentle ellipsoid	ellipsoid	sphere
	Method with no reduction of $\mathcal{Q}$				
20,000	7.96	253.78	5.50	9.02	3.53
40,000	7.37	435.14	5.72	9.89	3.56
60,000	7.06	526.74	5.88	10.39	3.61
80,000	6.89	591.28	5.99	10.68	3.66
100,000	6.79	663.68	6.10	10.41	3.61
1,000,000	5.38	1077.89	7.02	14.68	3.63
	Method with reduction of $\mathcal{Q}$				
20,000	7.81	22.68	5.59	8.60	3.75
40,000	7.33	19.81	5.79	9.56	3.68
60,000	6.90	26.97	5.85	10.35	3.65
80,000	6.90	27.29	6.00	10.75	3.75
100,000	6.75	22.78	6.11	11.07	3.65
1,000,000	5.07	10.70	6.63	12.82	3.61

Table 3 presents the performances (see section 7.2) of the ε-approximation of the diameter, without and with reduction of $\mathcal{Q}$, for various 3-D point distributions. It has to be compared to Table 2. The gain is especially tremendous for points distributed on a sphere, and is still noticeable for points distributed inside the sphere. CPU times for the same experiments can be found in Table 10.

7.5. Point sets in $\mathbb{R}^d$

Table 4 gives the performance measure (see section 7.2) of the exact computation of the diameter without and with reduction of the set $\mathcal{Q}$ for two dD points distributions (one being in convex position). As in 3-D for the same distributions, we only notice small differences between the computation of the exact diameter and its ε-approximation ($\varepsilon = 0.01$). CPU times for the same experiments can be found in Table 11.

We have observed that the complexity of computing a double normal is still a small constant, as in the 3-D case. Differently, the complexity of computing the diameter (*i.e.* the number of dot products) increases rapidly with d. This is due to the fact that the number of points in $\mathcal{Q}$ increases with the dimension. It can also be seen that the reduction of $\mathcal{Q}$ improves quite a lot the performances for high

Table 4. Performance measure of the computation of the exact diameter without and with reduction of $\mathcal{Q}$ for various d-D point distributions. Measures are averaged of several trials.

	in a cube			on a gentle ellipsoid		
n	6-D	9-D	12-D	6-D	9-D	12-D
	Method with no reduction of $\mathcal{Q}$					
20,000	72.39	447.02	1308.19	16.07	72.54	196.10
40,000	55.92	447.14	2233.32	18.72	66.43	221.37
60,000	39.17	643.96	2243.95	16.53	75.66	261.70
80,000	42.19	540.27	2365.45	13.82	76.36	325.04
100,000	46.62	732.92	2872.04	15.93	59.18	328.53
	Method with reduction of $\mathcal{Q}$					
20,000	17.70	48.52	204.23	10.36	18.45	35.08
40,000	15.75	51.07	208.53	11.43	20.37	44.12
60,000	15.72	43.55	264.78	10.73	18.67	33.82
80,000	15.98	45.26	225.18	10.96	16.16	41.66
100,000	16.18	35.07	260.83	10.95	16.99	39.10

dimensions.

7.6. Real 3-D objects

We present now results on real data from the Large Geometric Models Archive [b] in Table 5.

Table 5. Performance measures and CPU times for various models (averaged values over 100 trials).

Inputs	Bunny	Hand	Dragon	Buddha	Blade
Points	35,947	327,323	437,645	543,652	882,954
Exact computation of the diameter					
	Method with no reduction of $\mathcal{Q}$				
performance	1079.14	4.80	2121.64	2663.55	5.70
CPU time	5.86	0.23	139.27	188.04	0.68
	Method with reduction of $\mathcal{Q}$				
performance	1073.49	4.80	115.99	2395.42	5.70
CPU time	5.85	0.23	8.70	169.25	0.68
ε-approximation of the diameter ($\varepsilon = 0.01$)					
	Method with no reduction of $\mathcal{Q}$				
performance	816.57	4.80	1542.48	1377.36	5.70
CPU time	4.47	0.23	100.10	97.33	0.68
	Method with reduction of $\mathcal{Q}$				
performance	816.60	4.80	81.28	1233.47	5.70
CPU time	4.48	0.23	6.10	87.22	0.68

The results are quite disappointing. Our method is very sensitive to the "hardness" of computing the diameter of the given input, thus may yield large running

[b] "Large geometric models archive," `http://www.cs.gatech.edu/projects/large_models/`, Georgia Institute of Technology.

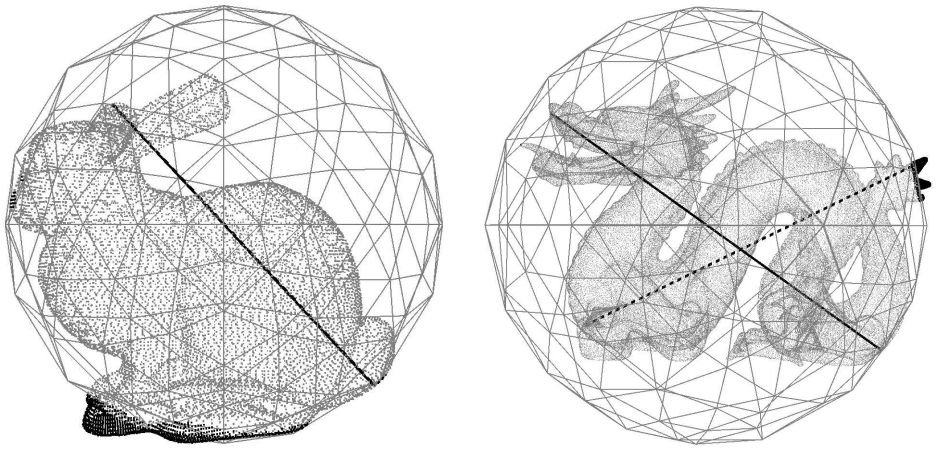

Fig. 3. Continous segment: the first computed double normal (in both cases, it's the diameter); dashed segment: the second computed double normal. In dark: the points remaining for the final part (the search over $\mathcal{Q} \times \mathcal{P}$), *i.e.* the points outside the sphere $B[pq]$ of diameter the best found double normal pq.

time even for small data sets. As already pointed out, high running time correspond to large sets $\mathcal{Q}$.

Let us have a closer look at the bunny and the dragon models. In both cases, the first found double normal is the diameter (the corresponding sphere is displayed in Figure 3), and a lot of points still remain in set $\mathcal{Q}$.

- For the bunny model, the second found pair was very close to the first one, thus the reduction of $\mathcal{Q}$ was not efficient. From the 1086 points of the first set $\mathcal{Q}$, only 7 have been removed. The same occurs for the Buddha model. This explains why there is a so little difference between our method with and without the reduction of $\mathcal{Q}$ for these two models (see Table 5).

- For the dragon model, the diameter goes from the nose to the middle of the last part of the tail. The second found pair goes from the very end of the tail to the throat. As it is very different from the first double normal, the reduction of $\mathcal{Q}$ was efficient in this case (see Table 5).

From the examples presented until now, it seems that the computation of the diameter pq can be very fast for the point distributions characterized by the fact that there are only a few points outside the closed ball of diameter pq (this is the case for both the hand and the blade models), but could be quite slow for other point distributions. It should be pointed out that most of the running time is spent on verifying whether the current diameter estimation is the true one or not.

8. Comparison with Har-Peled's method

The most comparable approach to ours is the one developed very recently by S. Har-Peled which exhibits good performances on most inputs, although it

is not worst-case optimal.[5] Har-Peled's algorithm is also able to compute a ε-approximation of the diameter. We briefly describe his method in section 8.1. Since the two methods are quite different and have different advantages and drawbacks, it is worth combining them, leading to good hybrid algorithms with more stable performances.

8.1. Har-Peled's method

In his approach, Har-Peled recursively computes pairs of axis parallel boxes (each enclosing a subset of the points). Differently from our method, Har-Peled's algorithm depends on the coordinate axes (see Table 7). The initial box is the one enclosing the original set $\mathcal{P}$, which means that the diameter is searched over $\mathcal{P} \times \mathcal{P}$. Then he recursively splits boxes along one axis. This way, the number of pairs of boxes increases but the average size of boxes decreases. Pairs that cannot contain a maximal pair are thrown away. The search over the pair $\mathcal{P} \times \mathcal{P}$ is then replaced by searches over a set of pairs $\{\mathcal{P}_i \times \mathcal{P}_j\}$.

To avoid maintaining too many pairs of boxes, Har-Peled does not decompose a pair of boxes if both contain less than $n_{\min}$ points (initially set to 40 in Har-Peled's implementation). Instead, he computes the diameter between the two corresponding subsets using the brute-force method. Moreover, if the number of pairs of boxes becomes too large during the computation (which may be due to a large number of points or to the high dimension of the embedding space), $n_{\min}$ can be doubled: however, doubling $n_{\min}$ increases the computing time.

We provide below an experimental comparison of both approaches, using the original Har-Peled's implementation[c] which only works for 3-D inputs. In order to be able to deal with inputs in higher dimensions, we have re-implemented his algorithm, following the same choices that were made in the original implementation.

8.2. Hybrid methods

It should be noticed that both methods can easily be modified to compute the diameter between two sets, *i.e.* the pair of maximal length with one point in the first set and the other in the second set.

Both methods have quadratic parts. Ours with the final computation over $\mathcal{Q} \times \mathcal{P}$, and Har-Peled's one when computing the diameter for a pair of *small* boxes.

We have implemented two hybrid methods that combines Har-Peled's method and ours. We first modified Har-Peled's algorithm by replacing each call to the brute-force algorithm by a call to our algorithm. We also tested another hybrid method where we modified our algorithm by replacing the final call to the brute-force algorithm (the search over $\mathcal{Q} \times \mathcal{P}$) by a call to the first hybrid method.

The two hybrid methods can be tuned by setting several parameters. The experimental results presented here have been obtained with the same values of the parameters.

The results show that the hybrid methods are never much worse than the best

[c] Available at `http://www.uiuc.edu/~sariel/papers/00/diameters/diam_prog.html`.

method. Moreover, their performances are more stable and less sensitive to the point distribution.

8.3. Experimental comparison of all methods

The performance measure designed in section 7.2 is not adapted for Har-Peled's method as it is not based on dot products as ours. Thus the comparison between the implemented methods is solely based on CPU times. It has only been done for the exact computation of the diameter.

Table 6. CPU times (seconds) on real inputs. HPM is short for Har-Peled's method.

Inputs	Bunny	Hand	Dragon	Buddha	Blade
Points	35,947	327,323	437,645	543,652	882,954
our method	5.73	0.29	8.51	172.91	0.49
HPM - original	0.08	0.45	0.90	0.72	1.00
HPM - our implementation	0.07	0.43	0.89	0.69	0.94
hybrid method #1	0.07	0.41	0.86	0.67	0.90
hybrid method #2	0.10	0.32	1.37	1.09	0.50

Table 6 presents the results on the real inputs. It can clearly be seen that Har-Peled's method generally outperforms ours on these inputs. Hybrid methods have comparable results to the best running time.

Table 7. CPU times for 3-D synthetic point distributions. The points sets distributed on the ellipsoid in the second and the third columns (bottom part of the Table) are identical up to a 3-D rotation.

	Volume					
Inputs	cube			ball		
Points	10^4	10^5	10^6	10^4	10^5	2.10^5
our method	0.01	0.19	0.53	0.04	0.79	1.20
HPM - original	0.01	0.18	1.96	0.31	18.16	53.88
HPM - our implementation	0.02	0.18	1.92	0.20	5.12	20.57
hybrid method #1	0.01	0.18	2.00	0.13	2.25	5.26
hybrid method #2	0.02	0.35	1.50	0.07	1.05	3.29
	Surface					
Inputs	ellipsoids			sphere		
	(gentle)		(rotated)			
Points	10^6	10^6	10^6	10^4	10^5	2.10^5
our method	1.34	2.02	1.61	1.08	358.21	-
HPM - original	1.78	3.84	37.70	2.13	95.49	328.90
HPM - our implementation	1.81	3.51	23.88	0.63	39.97	166.26
hybrid method #1	1.82	3.38	6.38	0.33	6.99	16.75
hybrid method #2	2.30	3.10	1.79	0.44	8.58	19.75

Table 7 presents results for various 3-D points distributions. It appears that our method performs a little bit better for points distributed inside a ball. It can also be seen that Har-Peled's method is very sensitive to the coordinate axes (because boxes are split along these axes). More interesting are the results for points distributed on a sphere: hybrid methods seem to outperform both Har-Peled's method and ours.

Table 8. CPU times for synthetic distributions ($n = 100,000$ points) in higher dimensions.

	Volume		Surface		
Inputs	cube	ball	gentle ellipsoid	ellipsoid	sphere
	6-D				
our method	0.31	36.95	0.11	0.33	-
HPM	0.85	466.44	0.97	0.87	465.08
hybrid method #1	0.67	77.20	0.79	0.73	118.06
hybrid method #2	0.66	63.31	0.19	0.65	142.38
	9-D				
our method	0.89	128.02	0.51	0.52	-
HPM	139.23	568.99	264.96	590.14	569.08
hybrid method #1	17.42	135.90	44.54	67.27	232.39
hybrid method #2	1.21	121.91	1.25	16.03	302.86
	12-D				
our method	3.87	445.03	1.08	7.88	-
HPM	629.37	651.56	648.88	650.98	647.74
hybrid method #1	44.45	354.14	58.53	56.11	511.41
hybrid method #2	19.72	380.41	13.00	24.62	745.60
	15-D				
our method	10.99	798.66	7.26	20.31	-
HPM	734.69	735.26	731.76	733.70	737.51
hybrid method #1	64.49	610.70	69.11	90.35	701.18
hybrid method #2	44.37	782.20	21.30	70.41	1120.57

Table 8 presents results for various d-D points distributions. Har-Peled's method appears to be generally less efficient than ours in high dimensions (except for distributions on the sphere).

From these results, it can be seen that hybrid methods represent a good combination of both methods. It should be pointed out that Har-Peled's method sorts points in boxes and, to do so, only requires to compare one coordinate per point. Hence its complexity is independent of the space dimension. Differently, our method uses dot products and its computational cost increases with the dimension. Nevertheless, contrary to our algorithm, Har-Peled's method has some parameters (see section 8.1) and implementation choices whose fine tuning may depend on the dimension (which we didn't do).

9. Conclusion

We have presented a very simple algorithm to compute the diameter of a set of points in any dimensions.

Our method is based on the computation of double normals. Computing a double normal appears to be extremely fast under any practical circumstances and in any dimension (despite the quadratic lower bound of lemma 4). Moreover, the reported double normal is very often the true maximal pair. This is not too much surprising since, on a generic surface, the number of double normals is finite and small. In any case, having a double normal provides a $\sqrt{3}$-approximation of the diameter in any dimensions.

However, even if the reported double normal pq is a maximal pair, it may be costly to verify that this is indeed the case. A favorable situation is encountered when the point set is contained in the ball B of diameter pq. The bad situation occurs when there are many points in set $\mathcal{P} \setminus B[pq]$ since we verify that none of these points is the endpoint of a maximal pair. This is the case for sets of constant width (e.g. a sphere) but also for some real models: e.g. the bunny, the dragon and Buddha (see Table 5).

Har-Peled's method does not suffer from this drawback. However, it depends on the coordinate axes (since the boxes are aligned with the axes) and gets generally worse than ours with increasing dimension.

To combine the advantages of both approaches, we designed two hybrid methods which appear to be efficient in all cases.

The source code of the program used in the experiments is also available [d].

References

1. N. M. Amato, M. T. Goodrich, and E. A. Ramos, "Parallel algorithms for higher-dimensional convex hulls," *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, 1994, pp. 683–694.
2. S. Bespamyatnikh, "An efficient algorithm for the three-dimensional diameter problem," *Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1998, pp. 137–146.
3. Otfried Cheong, Chan-Su Shin, and Antoine Vigneron, "Computing farthest neighbors on a convex polytope," *Proc. 7th. Annu. Int. Comput. and Combinat. Conf.*, 2001, volume 2108 of *LNCS*, pp. 159–169.
4. K. L. Clarkson and P. W. Shor, "Applications of random sampling in computational geometry," *Discrete Comput. Geom.*, **4** (1989) 387–421.
5. S. Har-Peled, "A practical approach for computing the diameter of a point-set," *Symposium on Computational Geometry (SOCG'2001)*, 2001, pp. 177–186.
6. Grégoire Malandain and Jean-Daniel Boissonnat, "Computing the diameter of a point set," Research report RR-4233, INRIA, Sophia-Antipolis, 2001, `http://www.inria.fr/rrrt/rr-4233.html`.
7. F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, 3rd edition, (Springer Verlag, October 1990).
8. E. Ramos, "Intersection of unit-balls and diameter of a point set in R^3," *Comput. Geom. Theory Application*, **8** (1997) 57–65.
9. E. Ramos, "Construction of 1-d lower envelopes and applications," *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997, pp. 57–66.
10. Edgar A. Ramos, "An optimal deterministic algorithm for computing the diameter of a three-dimensional point set," *Discrete & Computational Geometry*, **26** (2001) 245–265.
11. A. C. Yao, "On constructing minimum spanning trees in k-dimensional spaces and related problems," *SIAM J. Comput.*, **11** (1982) 721–736.

[d] "Source code of program for computing or approximating the diameter of a point set in d-d," `http://www-sop.inria.fr/epidaure/personnel/malandain/diameter/`

Table 9. CPU times (in seconds) of the computation of the exact diameter without and with reduction of $\mathcal{Q}$ compared to the brute-force algorithm for various 3-D point distributions. Measures are averaged of several trials.

	3D Points distribution					
	Volume		Surface			
n	cube	sphere	gentle ellipsoid	ellipsoid	sphere	cube
	Method with no reduction of $\mathcal{Q}$					Brute force
20,000	0.026	1.29	0.017	0.064	21.9	20.9
40,000	0.051	4.40	0.034	0.156	95.0	92.9
60,000	0.077	9.43	0.052	0.314	215.3	215.0
80,000	0.102	16.04	0.071	0.369	384.0	384.5
100,000	0.123	21.92	0.093	0.345	601.6	602.0
1,000,000	1.253	-	0.999	5.904	-	-
	Method with reduction of $\mathcal{Q}$					
20,000	0.024	0.145	0.017	0.031	12.4	
40,000	0.048	0.398	0.034	0.071	60.8	
60,000	0.074	0.641	0.053	0.101	137.3	
80,000	0.095	1.023	0.072	0.140	252.5	
100,000	0.121	1.344	0.089	0.182	397.1	
1,000,000	1.210	-	1.002	1.757	-	

Table 10. CPU times (in seconds) of the computation of the ε-approximation of the diameter without and with reduction of $\mathcal{Q}$, $\varepsilon = 0.01$.

	3D Points distribution				
	Volume		Surface		
n	cube	sphere	reg. ellipsoid	ellipsoid	sphere
	Method with no reduction of $\mathcal{Q}$				
20,000	0.020	0.630	0.014	0.021	0.009
40,000	0.038	1.842	0.028	0.047	0.019
60,000	0.055	4.247	0.044	0.074	0.028
80,000	0.070	5.863	0.059	0.102	0.037
100,000	0.086	8.057	0.075	0.126	0.046
1,000,000	0.638	170.620	0.869	1.848	0.454
	Method with reduction of $\mathcal{Q}$				
20,000	0.020	0.056	0.014	0.021	0.009
40,000	0.038	0.114	0.028	0.046	0.018
60,000	0.055	0.176	0.043	0.074	0.027
80,000	0.072	0.198	0.059	0.104	0.036
100,000	0.085	0.234	0.074	0.130	0.047
1,000,000	0.667	1.395	0.843	1.688	0.449

Table 11. CPU times (in seconds) of the computation of the exact diameter for various d-D point distributions.

	in a cube			onto a regular ellipsoid		
n	6-D	9-D	12-D	6-D	9-D	12-D
	Exact diameter, method with no reduction of $\mathcal{Q}$					
20,000	0.204	2.290	10.66	0.075	0.330	1.55
40,000	0.465	5.901	26.32	0.142	0.709	3.06
60,000	0.562	8.238	54.67	0.224	1.473	5.33
80,000	0.937	13.291	83.93	0.291	1.582	5.75
100,000	0.910	18.427	113.11	0.345	2.154	10.85
	Exact diameter, method with reduction of $\mathcal{Q}$					
20,000	0.079	0.293	1.469	0.047	0.108	0.368
40,000	0.154	0.575	3.699	0.088	0.235	0.562
60,000	0.215	0.701	5.158	0.140	0.333	0.761
80,000	0.280	0.878	7.393	0.188	0.445	0.744
100,000	0.383	1.136	7.449	0.241	0.487	1.101